PATENT
Attorney Docket No. PD98-2385/34309.0006
Express Mail No. EL353030753US

SYSTEM AND METHOD FOR SECURELY DOWNLOADING FIRMWARE TO STORAGE DEVICES AND MANAGING DEVICES IN A CLIENT-SERVER ARCHITECTURE

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates in general to managing storage devices in a networked computer system. More particularly, this invention relates to a system and method for securely downloading firmware to storage devices and managing storage devices in an integrated storage management client-server environment.

2. Background

Computer systems, sometime called hosts, have secondary storage devices such as hard disk drives, tape drives, optical disk drives, or other persistent storage devices, attached to the computer system directly through an adapter or controller, or attached over a network. A system administrator of the computer system has the responsibility to manage the storage devices whether the storage devices are attached to the host through an adapter, controller, or via a network. Identifying the status of various devices, downloading firmware to the controller and storage devices, handling device mode pages and log pages, and monitoring device status are a few of the many storage management tasks that an administrator has to perform on a daily

basis. An administrative tool, including a software tool, is generally required to manage the storage subsystems.

Conventional software products for storage device administration generally permit firmware to be downloaded to storage devices which are directly attached to a host, and runs as an application where the storage device is connected. However, these software products generally do not operate over a network as an integral part of a storage management tool, and they generally permit firmware to be downloaded to only one device at a time. As a consequence, the firmware file to be downloaded must exist in the local computing system where the storage device is physically located. Further, some of the conventional storage device administration software does not support device specific variable length segmented firmware downloading. Finally, some of the conventional storage device administration software tools generally do not provide security for the firmware download.

It is with these shortcomings of the existing art in mind that the present invention was developed. What is needed is a system and method for securely downloading firmware to storage devices and managing storage devices over a network in a client-server architecture.

SUMMARY OF THE INVENTION

In light of the above, therefore, in one embodiment of the present invention, a method for downloading a firmware file to a storage device of a computing system having a client connected to a server over a network is disclosed. The method includes a step of creating a firmware file containing a

firmware image with a data header embedded therewith. A transferring step transfers the firmware file over the network from the client to the server, and an instructing step instructs an agent operating on the server to download the firmware image to the storage device. In responsive to the instructing step, an initiating step initiates a process or thread for downloading the firmware image to the storage device.

The data header includes data corresponding to identification of the storage device type, data corresponding to an encrypted password associated with the firmware file, and other data fields.

In this manner, embodiments of the invention permit downloading firmware to storage devices, such as a disk or array of disks, in a secure method over a network. The storage devices can be configured, and managed, from one or more management stations over a network.

In another embodiment of the invention, a system for managing storage devices in a computing system having a client connected to a server over a network is disclosed. The system includes one or more agents, an optional agent manager, one or more applets, and an applet manager. Each agent operates on the server and communicates with a storage device, while the agent manager manages the agents. Each applet operates on the client and is adapted to provide a user interface for managing the storage devices. The applet communicates with the agent over the network to obtain information about the storage devices. The applet manager running on the client manages and communicates with each applet. The system can further include a storage

management database (in memory or persistent) maintained by the agent containing data about the storage devices.

The storage devices are represented as objects instantiated in the agent and applet. The agent is a process running on the server. It includes a multiprotocol layer communicating with the storage devices, an object layer representing the storage devices, a command layer adapted to support adapter/controller specific commands to manage the storage devices, and a multiprotocol network layer.

The agent registers with the agent manager when it starts. When the applet wants to talk to the agent, first it contacts the agent manager to obtain information about the agent it wants to communicate therewith. The agent manager passes identification and connection information about the requested agent to the applet only after performing a security authorization check on the applet. To improve reliability, the agent manager optionally can poll the agents to determine if the agents continue to operate on the server. When an agent terminates, it un-registers with the agent manager, and the services of the agent are unavailable thereafter.

The applet is designed in a modular and object oriented fashion with interfaces such as an API between them. One of the modules contains objects of the communication and storage subsystems including devices and adapters/controller. Before the applet can communicate with the agent, the applet must provide the identification information to the agent manager, and the agent manager authenticates the applet. In configurations where the agent

manager is not present, the agent will perform authentication. The applet can access the data about the storage devices by generating requests to the agent.

In order to download a firmware image to a storage device in the system over the network, a firmware file is created which is capable of supporting multiple storage devices with varying characteristics. The firmware file includes a copy of the firmware image with a data header embedded in the firmware file. The header further includes data corresponding to revision numbers, data corresponding to identification of the storage device type, and data corresponding to an encrypted password associated with the firmware image, among other things.

When the user issues a firmware download command, the applet requests the firmware filename and password to ensure that the user is authorized and is the right firmware image for the specified device(s). After validation, the applet transfers over the network the firmware file to the agent, then instructs the agent to download the firmware image to one or more storage devices, and in response, the agent initiates one or more processes or threads for downloading the firmware image to the storage devices in a secure manner.

The system also provides distributed, load sharing, robust asynchronous event notification, and multilevel security mechanisms for storage device management.

The foregoing and other features, utilities and advantages of the invention will be apparent from the following more particular description of a

preferred embodiment of the invention as illustrated in the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 illustrates a bock diagram of a device management system in a client-server architecture in accordance with the present invention.

Fig. 2 illustrates a modular and object oriented block diagram of the device management architecture of the client side applet in accordance with the present invention.

Fig. 3 illustrates a block diagram of the server side architecture in accordance with the present invention.

Fig. 4 illustrates a block diagram of the layers of an agent shown in Fig. 3, in accordance with the present invention.

Fig. 5 illustrates a data structure for a peripheral master configuration header in accordance with one embodiment of the present invention.

Fig. 6 illustrates a data structure of a command array processor data of group 0 for end of command markers, in accordance with one embodiment of the present invention.

Fig. 7 illustrates a data structure of command array processor data of group 1 for SCSI commands, in accordance with one embodiment of the present invention.

Fig. 8 illustrates a data structure of command array processor data of group 2 for custom commands and handling descriptors, in accordance with one embodiment of the present invention.

Fig. 9 illustrates a data structure of command array processor data of group 3 for screen commands, in accordance with one embodiment of the present invention.

Fig. 10 illustrates a flow diagram of the steps for downloading firmware using the data structure for a peripheral master configuration header, in accordance with the present invention.

Fig. 11 illustrates a data structure for a peripheral simple configuration header in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

In accordance with the present invention, a device management system is provided for downloading firmware, configuring, and managing one or more storage devices over a network using the client-server architecture. As will be discussed below, the system 20 of Fig. 1, which is based on the client-server architecture, permits simultaneous and secure downloading of firmware to the storage devices specified by the user. The architecture supports multiple interfaces, multiple command sets, multiple protocols, and multiple hosts with multiple storage subsystems.

As described below, the present invention discloses a novel format for a firmware file suitable for secure firmware downloading. The firmware file has a copy or image of the firmware, along with a unique data header.

The system 20 provides asynchronous event notification services (AES) using distributed AES servers. An AES server is capable of running on a client

station 42, on an independent system 44 in the network, or on a server system 50 as shown in Fig. 1.

The TCP/IP protocol is used to communicate between the server and the client over the network, although it is possible to use other network protocols. Where there is no network, the client is capable of communicating with the adapter or controller coupled to the storage devices using either a SCSI interface or serial port, as shown in Fig. 1 by client 40 coupled to server 22.

As shown in Fig. 1, the system 20 includes a server component 22 and a client component 24 communicating over a network 25. The server component 22 has a storage subsystem 26 having, for example, a plurality or array of SCSI devices 28 coupled to an adapter 32 or controller 30, and the controller, in certain configuration, interfacing directly or through an adapter 32 coupled to the server host. A system 50 can have one or more storage subsystems, as shown in Fig. 1 as 52 and 54.

In accordance with the present invention, the server component 22 is provided with an agent 34 operating on the server, and an optional agent manager 36. In accordance with the present invention, each storage subsystem has an agent that is responsible for controlling and configuring the storage subsystem. Each agent communicates with the storage devices of the subsystem through the controller/adapter. In one embodiment of the present invention, server 50 has agent 56 controlling storage subsystem 52 and agent 58 controlling storage subsystem 54.

The agent manager 36 manages the agents, as will be described below. One of the functions of the agent manager is security, that is, to ensure that only authorized users can access and manage the storage subsystems to perform administrative operations. In one example, the agent manager 60 manages the agents 56 and 58 and provides a secure interface from any client applet wishing to communicate with agents 56 or 58. Depending on the particular implementation, where there is no agent manager, the functions and operations performed by the agent manager are incorporated in and performed by the agents.

In accordance with the present invention, the client component 24 runs on a computing system that is designated as the management station. It has an applet manager and one or more applets, wherein each applet is adapted to interact with an agent to manage the storage devices in the subsystem. The applet manager manages all of the applets including launching the applets, and passing appropriate information to the applets. The applet manager and the applets provide a user interface to manage the storage devices in the subsystems. Generally, the client component supports multiple user interfaces such as GUIs (Graphical User Interfaces), and command interfaces.

The client component also supports different types of controllers and adapters by having different command sets to manage the storage devices coupled thereto. Different applet uses different command set to communicate with the corresponding agent based on the type of controller used. It is understood that the type of controller or adapter used is a matter of choice

depending on the particular implementation in which the present invention operates within.

Storage specific features are contained within the pair of applet and agent corresponding to a storage subsystem. The architecture is modular and permits dynamic addition and deletion of applets and agents as subsystems are added and deleted. Thus, management functions can be dynamically added or removed in accordance with the dynamic changes to the storage subsystems.

In one embodiment of the invention, the various components of the device management architecture are implemented as modules and each module consists of objects in an object-oriented model. According to one embodiment of the present invention, and as shown in Fig. 2, each applet consists three different modules, an adapter/Controller GUI module 110, a device GUI module 112, and a storage subsystem module 114.

The adapter/controller GUI module 110 is the main software module of the applet, and performs such functions as displaying a list of storage devices and their statuses, coordinates the download of firmware using other modules, and service traps. In order to ensure that the device management functions and the code is usable across multiple adapters and controllers, the GUI functions related to device management are kept separately in the device GUI module 112. For example, the device GUI module 112 displays properties and settings for a selected device, and initiates the process for downloading firmware with proper security verification.

The subsystem module 114 reflects the network connection, the adapter/controller, and various storage devices at the agent in the form of objects. The subsystem module, in addition to creating objects representing devices, adapter/controller, and network, is capable of building and transmitting commands to the agent and receive responses for them from the agent. It can do so at the request of other modules in the applet or on its own depending in the condition of the applet.

The applet includes several objects that provide methods for performing various functions. Fig. 2 shows a device GUI interface object 120, subsystem device objects 122, an adapter/controller object 124, and a network object 126. The adapter/controller GUI module 110 communicates with the device GUI module 112 through the device GUI interface object 120 API. It 110 also communicates with the subsystem module 114 using the methods of the subsystem device objects 122 and the adapter/controller object 124. Certain member functions of some of the subsystem device objects 122 and the adapter/controller object 124 that need to communicate to the agent do so using the network object 126.

Using a class hierarchy, the various objects are instantiated from their classes. In one example of the embodiment, there are three classes: (1) a device base class, (2) a device subsystem class, and (3) a device GUI interface class. The device subsystem class is derived from the device base class. The implementation of some of the device subsystem class member functions will be different between adapters and between controllers depending on how the

adapters/controllers communicate with the agent. It is also possible that the device member functions could have different private data members.

With regard to the functionality of the objects, the member functions in the base class, in one example of the embodiment, do not communicate with the agent. Since the commands and formats used to communicate with the agent differ between controllers and hence their implementation will be different between controllers, member functions that need to communicate with the agent are specified as virtual in the base class. The virtual functions in the device base class should be implemented in the classes derived from the base class. Their implementation will differ depending on the controllers and adapters used.

The information in the device classes include data such as device name, global ID, capacity, state, vendor ID, product ID, revision, vendor specific information, serial number, mode pages, and channel.

In operation, there is one object for each adapter/controller in the applet. For each device connected to the adapter/controller, there is a device object. When a user wants to see the properties of a device or wants to download firmware to a device, the applet creates a device GUI interface object and associates it with the corresponding device object, and activates the device GUI module passing appropriate information through the device GUI interface object. When the user exits the device GUI, the device GUI interface object goes out of scope.

In one embodiment, the applet has several layers for interfacing and passing data. At the top layer, the applet interfaces with the applet manager and the user. At the bottom layer, the applet is capable of supporting multiple protocols including TCP/IP. In the middle layers, there is subsystem object layer and command layer. The subsystem object layer interfaces with and maintains the managed storage objects such as devices, adapters, etc. The command layer contains commands which depend upon the controller or adapter types supported by the applet.

Fig. 3 illustrates a block diagram of the server side architecture in accordance with the present invention. Depending on the number of storage subsystems in a host computing system, there could be zero or more agents running at the server host. As shown in Fig. 3, the server component has an agent manager 106 responsible for all the agents 100, 102 running at the server host. The agent manager manages the agents running on the host.

A storage management database 104 is shown for maintaining data relating to security and other data relating to the agents and storage subsystems. It is possible to keep the database either in memory or on persistent storage.

In one embodiment of the inventions, the agent registration, and agent un-registration features are provided for the agent manager to track the currently running agents. When an agent starts, it registers with the agent manager. As part of the registration, the agent passes information that uniquely identifies the agent, subsystem, as well as, information necessary for a

client to establish a connection to the agent. When the agent shuts down or is brought down by the user, it notifies the agent manager to un-register it. Upon receiving the request to un-register the agent, the agent manager removes the agent information from the lists of agents being managed.

The Agent registration and un-registration feature permits the architectures to be scalable by dynamically providing management support to newly added subsystems and removing support to subsystems that are removed from the host.

In one example, the agent manager uses a known port to service requests from clients. The agents do not use well-known ports to communicate with clients. The addresses of agents are obtained through the agent manager. The advantages are all clients need to know only one port, which is the well-known port of the agent manager, and there are savings of well-known ports which are limited. There is no need to make the ports all agents as well-known ports.

In accordance with one aspect of the present invention, a protocol for connecting a client to an agent is provided. Any client (*i.e.*, applet) wanting to establish a connection to an agent must contact the agent manager first with proper agent identification and client authentication. The agent manager, by using authentication data in its security database, first authenticates the client to ensure that it is a valid client. If the client is authorized to communicate with the agent and if the requested agent is running, the agent manager passes the connection information of the agent to the client. If the client is not

authorized to communicate with the agent, or if the requested agent is not running, the agent manager will deny the connection request. Thus, agent manager provides access security to its registered agents. The client establishes a connection to the agent using the information received from the agent manager. After establishing connection to the agent, the applet directly communicates with the agent without involving the agent manager.

After registering with the agent manager, it is possible that an agent may fail or terminate without un-registering with the agent manager. To improve reliability, optionally, the agent manager can periodically ping or poll the agents for their current status to ensure that the registered agents are running properly and have not terminated. If an agent does not respond to the pings for a certain number of times, the agent manager un-registers the agent and changes its status to not available.

Storage management information is maintained in database(s) 104 as shown in Fig. 3. The databases can be kept in memory or on persistent storage as needed. Information that is not appropriate to be kept in the database is obtained in real-time from the subsystem and passed to the client. Both the agent manager 106 and agents 100, 102 can access the appropriate part of the database(s) and extract necessary information, as needed. The access to the database can be centralized or distributed.

Normally, an applet communicates with the agent to get information about storage subsystem part or all of which could be in the database or may have to be obtained in real-time from the subsystem. The agent will get the

information from the appropriate place(s) and pass it to the applet. The concept of applet-agent communicating with each other in managing storage subsystems makes the architecture modular. Additionally, it allows different protocol layers and interfaces to be used between different applets and the corresponding agents.

Alternatively, the client can pass a request to the agent manager for information kept in the database. The advantage of obtaining data through the agent manager is that a single point of contact is provided to the clients, as opposed to every client having necessary protocol stack for interfacing with the agents. Also, information about various subsystems can be maintained in a uniform way. The agent manager can further provide necessary security mechanism in accessing the information from a single point, as described above. On the other hand, the disadvantage of the centralized access is that the agent manager may become a bottleneck; also for information that are required to be obtained in real-time, the agent manager has to request the agent to get it from the subsystem and store it in the database. If the organization of information about various subsystems differs between agents, then the agent manager has to know how to access several databases.

Fig. 4 illustrates a block diagram of the layers of an agent, in accordance with the present invention. The device interface layer 206 communicates with the storage subsystem 202 either directly using its sub-layer 208 or through the device driver 200 using its sub-layer 204. The managed subsystem object layer 210 has various objects that represent the

subsystem managed by the agent. This layer 210 communicates with the device interface layer 206 to obtain information necessary to populate the objects. It is the counterpart of the applet subsystem managed object layer. The managed subsystem layer uses the command layer 212 to receives various commands from the applet and send responses back to the applet. Invariably, different subsystems use different command sets with varying command structures. The command layer is capable of supporting multiple command sets according to the subsystems. The command layer in turn uses the network layer 214 to communicate with the applet. The network layer 214 is capable of supporting multiple protocols.

Accordingly, with the use of the device management system shown and described, various features, operations, and functions can be performed to manage and configure the storage devices over a network.

FIRMWARE DOWNLOAD

Downloading firmware in a secure manner to the devices in one or more subsystems and one or more hosts is an important function of storage subsystem device management. The firmware download operation, also referred herein as "codeload", should be a secure operation since failure or mismanagement can result in permanent device failure and loss of data on the device.

In accordance with the present invention, the device management system shown in Fig. 1 can be used for downloading firmware to one or more storage devices. The firmware to be downloaded to the storage devices

resides, in one example, at the client management station. In accordance with the present invention, the firmware file, herein known as the ASCII Codeload Image File (ACIF) contains a firmware image, and a data header. As used herein, the term firmware image means a copy of the firmware of the storage device. The firmware download process is accomplished in two phases. In the first phase, the firmware file is transferred from the applet to the agent. In the second phase, the agent is instructed by the applet to load the firmware, (*i.e.*, the firmware image), to the storage device.

In one example, each firmware image is preprocessed and a new firmware file is created with header information added to or embedded to the original image. The header provides important information about the firmware and permits version checking which reduces the chance of inadvertently downloading a wrong firmware version to a device. Two forms of data structures containing the header information and the firmware image are shown in Figs. 5 and 11 in accordance with the present invention. The header format disclosed herein is capable of supporting several devices with specific characteristics.

Fig 5 illustrates a data structure 300 for the ASCII Codeload Image File (ACIF), in accordance with one embodiment of the present invention. The ACIF is divided into two discrete elements: a Peripheral Master Configuration Header (PMCH), and the firmware image used by the device(s). The ACIF structure allows for support of multiple device types that utilize the same firmware image. The PMCH, as part of the ACIF structure, contains various

data fields relative to the device update process that define the supported device(s) and their configuration settings.

An ACIF File Checksum field 302 contains the checksum of the ACIF file in its entirety. An optional Filename field 304 is provided for reference as an aid to the user and/or the application for file verification purposes. A Preprocessor Revision field 306 is provided to document the revision number of the preprocessor editor that created the ACIF, in the event of an incompatibility between the ACIF structure and the system-supported, such as the addition of features provided in the PMCH. An ECO field 308 can be used to provide an engineering change order (ECO) number, as a method of tracking device updates. To prevent unauthorized use, an Encrypted Password field 310 is provided which allows the device update if a password is verified.

A group of fields 312 & 314 are provided in the PMCH to identify 'N' possible devices that use the same firmware image. In Fig 5, six (6) fields are shown for each group 312 and 314, however additional fields could be added as needed. These fields are pointers into a data storage area 320 that contains both INQUIRY and MODE data that identify and validate the device(s) for update. These groups of fields, independent of each other, identify each of the 'N' supported devices. As shown in 312 and 314, the MODE fields may be used to specify OLD and NEW settings for two of four possible MODE data types, the Default MODE as well as the Saved MODE settings. If desired, additional fields could be added to support the remaining two MODE types,

Changeable and Current. The applet could, if desired, make use of these MODE fields to configure a device.

For each group of fields 312 and 314 specifying a device, a CAP Instruction Sequence field 316 is provided which defines the sequence of operations to be performed to the specific device during the codeload and/or MODE update processes. An End-Of-List marker 318 signifies the end of the device information.

A Storage Area field 320 contains the new and old INQUIRY and MODE data for all supported devices. The respective pointer fields for each device, as detailed above, reference this storage area in the device update process. Following this data area is another End-Of-List marker 322 to signify the end of the data storage area field.

The Encrypted Binary Firmware Image field 324 contains the new firmware image, as provided by the device manufacturer, which is to be loaded into the device. For security purposes, an encryption algorithm may be used to protect image tampering. Finally, a Firmware Image Checksum field 326 is provided to verify the integrity of the image before the code load operation.

It is understood that the arrangement of the master configuration header is a matter of choice and could be varied within a particular application without departing from the spirit of the invention described herein.

The CAP variable definitions are categorized into four groups including an end of command marker, a set of SCSI commands, a set of custom commands and handling descriptors, and a set of screen commands for passing

data to the user. These CAP commands are shown in Figs. 6-9. It is understood that the CAP commands shown and described herein are by way of example only, and the name or value assigned to each command is matter of choice depending on the particular implementation.

Fig. 10 illustrates a flow diagram of the steps for downloading firmware using the data structure for a peripheral master configuration header, in accordance with one embodiment the present invention. Operation 1000, which is performed by the agent, scans the peripheral bus to identify storage devices connected to the storage subsystem. This information is transferred to the applet at its request. The storage devices located are displayed to the user for selection. In operation 1002, the user selects a list of devices to update. Operation 1004 queries the user for a password. The password supplied by the user is compared with the password in the ACIF file to insure that a non-authorized user cannot access and manipulate the storage device. Operation 1006 determines if the password is correct, and if not, access is denied and control is passed to the end of the process 1040.

If the user provides the proper password, operation 1006 passes control to operation 1010. Operation 1010 retrieves the Inquiry string from the file header for device validation. Operation 1012 retrieves the device Inquiry and Mode data from all selected devices through the agent. Operation 1014 then compares the Inquiry string from the image file with the Inquiry data from the selected devices.

Decision operation 1016 then determines if all selected devices are updateable. If true, control passes to operation 1030. If there are any non-updateable devices in the list, then control is passed to operation 1018, which displays the list of invalid devices to the user. In operation 1020, the use removes the invalid devices from the list of devices selected. Operation 1022 determines if there are any updateable remaining devices in the list. If the list is non-empty as determined by Operation 1022, then control is passed to Operation 1030. If the list is empty, then control is passed to End-Of-Process 1040.

Since both the validity of the user and the validity of the particular storage device(s) have been confirmed by operations 1006 and 1016, operation 1030 transfers the firmware file to the agent(s). Operation 1032 instructs the agent(s) to download the firmware to the particular storage device(s). An embodiment of this invention, supports a 'segmented-download' feature.

A segmented-download occurs by transmitting the firmware image to the device in fixed-length sections (for example, 32K) using a series of Write Buffer commands which define both the transmission length and offset into the firmware image. The segment size may vary between devices. Using this method, the agent transmits the file segments to all selected devices. By doing parallel transmission, the devices(s) receive the final Write Buffer segments and perform the re-programming process simultaneously, thus reducing the overall maintenance time. In large disk arrays, this time could be significantly large. In one embodiment of this invention, what would have taken several

minutes, or perhaps hours, to perform on a large disk array update would be reduced to just a few minutes.

The update process is controlled through the use of the CAP commands and instructions present in the master configuration header. The storage device is locked during firmware download preventing other applications from using it. Following the updates, operation 1036 rescans the peripheral bus to get the new Inquiry data from the devices. The new Inquiry data is compared against the new Inquiry strings from the master configuration header 316 for update verification. Thus, Fig. 10 illustrates the secure downloading of firmware to devices over a network in a client / server architecture.

Fig. 11 illustrates an alternative data structure for a configuration header in accordance with another embodiment of the present invention. The header 1100 shown in Fig. 11 has fewer data fields than the header shown in Fig. 5. Referring to Fig. 11, the header 1100 has a field 1102 for the header size, which is the size of the header itself. Fields for the revision number 1104 of the header, the vendor identification 1106, and the product identification 1108 are provided in the header. The field FW Rev 1110 indicates the revision of the firmware image.

The encrypted password 1112 is specific to the firmware file. The encrypted password allows the client to ensure that unauthorized users will not be allowed to download the firmware. The agent compares the existing version of the firmware on the device and the version to be downloaded. Appropriate warning is generated to the user if a discrepancy is identified.

The comment field 1114 is used to convey information that will be useful to the user at firmware load time. The firmware supplier/distributor specifies the information that goes into the comment field at preprocessing time. The FW length field 1116 specifies the size of the firmware image.

Finally, the firmware image 1118 is attached to the header. The firmware image will be used as needed when a codeload is specified. Optionally, a checksum could follow the firmware image field to ensure proper reception of the file without any transmission errors.

In a small computing system where there are only few storage devices, updating the device firmware could be done one at a time. However, in larger computing environment, there are many storage devices of the same type in one or more subsystems connected to a host, and often distributed amongst multiple systems. In accordance with the present invention, the user can select multiple devices of the same type to perform firmware download simultaneously. . The selected devices could be in one subsystem or across multiple subsystems within a host or amongst multiple hosts. The agent spawns multiple threads, one for each storage device, and loads the firmware simultaneously.

In the case of multiple simultaneous downloads, the file can be sent from the applet to the agent in various ways. In one example, the firmware file is transferred from the applet to the agent once for each storage device. This is a time consuming process. In a second example, the firmware file is transferred from the applet to the agent once, and the agent is instructed to

load the firmware on multiple storage devices by providing a list of storage devices that are managed by that agent.

In a third example, if the storage devices are spread across multiple agents in a host, the file transfer from the applet to the agent is still performed once and stored on the host. Then each agent is provided with a list of storage devices managed by it, and requested to perform firmware download using the firmware file that was previously transferred.

In a fourth example, where firmware download has to be performed across multiple hosts, the steps of the third example can be used once for each host in a serial fashion. Alternatively, the firmware file can be transferred once by multicasting it to the set of hosts. Then the agents are provided with a list of storage devices to perform the firmware download.

To ensure that the firmware file has been successfully received, the agent checks the byte count of the received file with the byte count of the original file. Where there is checksum, the agent performs checksum to ensure error free reception of the file.

The agent communicates with the clients while the firmware download is in progress. To ensure that the download is not interrupted due to network problems, the agent creates a separate thread (or process) to download the firmware to the storage device. In one example of the present invention, the main agent thread (or process) communicates with the client, and the download thread (or process) and the main agent thread (or process) communicate through interthread (or interprocess) communication mechanism.

While firmware download is in progress, termination of the agent and or the firmware download process will render the storage device unusable. In order to prevent such a mistake, the agent catches termination signals and warns the user appropriately, thus improving firmware download reliability. Thus, the thread-per-download coupled with signal handling and validation enables secure reliable simultaneous firmware download with high performance.

MULTI-LEVEL SECURITY

As discussed above, administering storage device should be a secure operation in order to ensure that unauthorized users do not access the storage subsystem. In accordance with the present invention, security can be provided at multiple levels, including (1) system-level authentication, (2) multi-level password security; (3) user-level authentication, (4) message-level authentication, and (5) encryption.

With system-level authentication, a security check is performed at the system level. The agent manager maintains the list of client systems that are authorized to establish communication with the agents running on its host. When a connection request is received from a client, the agent manager checks its security database to see whether the client system is an authorized system. The agent manager refuses to permit unauthorized clients to establish connection with its agents.

With multi-level password security, the storage administrative functions are classified into various sets of functions. Each set of functions is assigned

different password level. For example, some basic monitoring operations may not have any password at all. On the other hand, operations such as firmware download are classified to have highest level of password protection. Based on the operation performed, the user must specify the appropriate level of password.

With user-level authentication, various levels of privileges are created, and each user is assigned a privilege level. Based on the level of privilege, users perform different level of management functions without the need to specify multi-level passwords described earlier. In other words, when the users log into the management station, they automatically get the privilege level assigned to them. A lower privileged user could perform upper level functions by going through the multi-level password security mechanism described above.

With message level authentication, the client includes its authentication information with each message sent to the server. One disadvantage with this scheme is that processing the authentication information requires extra computing time. As a compromise, the authentication information could be included on selected message types only, instead of all messages.

With encryption, information exchanged between the client and the server is encrypted. Again to minimize processing overhead, selected information could be encrypted.

ASYNCHRONOUS NOTIFICATION

The status of a storage device can change at any given time. For example, a normally operating disk can fail suddenly, or a fan may stop running and the temperature of a cabinet may rise beyond the critical limit. It is important that the administrator is notified of the status changes in the subsystems immediately so that appropriate action can be taken before major failures occur. In accordance with one embodiment of the invention, when an agent detects an important status change in a component of a storage subsystem, the agent sends an event notification message (trap) to one or more designated asynchronous event servers (AES). The AES in turn sends messages (traps) to the applet manager, to the appropriate applet, and or to an external entity. In addition, the AES delivers a message to the designated user. In one example, the message is in the form of a pager message and/or an electronic mail message, marked urgent if needed.

One or more AES servers run in the networked environment. In one example of the embodiment, the notification workload is distributed among multiple AESs. In another embodiment, one AES acts as the primary AES and handle all notifications. In this case, other AESs, designated as secondary AESs, are in the standby mode. When the primary AES fails, any one of the secondary AESs will become the primary AES after performing arbitration among secondary AESs. To keep all AESs synchronized, the notifications are sent from the agents in a multi-cast mode or the AESs exchange information among themselves. Broadcast mechanism is also possible, however, it has less

security since anyone listening on the network can get it. The primary AES notifies its presence by transmitting an "alive" message periodically to the secondary AESs.

The distributed AESs technique provides better performance by sharing the workload. The primary-secondary AES mechanism provides enhanced reliability since any AES can become a master by arbitration when an existing master AES fails. With persistent data storage at the AES and the retry (instead of one-time) feature increases the reliability of notification.

While the invention has been particularly shown and described with reference to a preferred embodiment thereof, it will be understood by those skilled in the art that various other changes in the form and details may be made without departing from the spirit and scope of the invention.